Problem 6 (30 points)

Problem Description

In this problem you will train decision tree and random forest models using sklearn on a real world dataset. The dataset is the *Cylinder Bands Data Set* from the UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Cylinder+Bands. The dataset is generated from rotogravure printers, with 39 unique features, and a binary classification label for each sample. The class is either 0, for 'band' or 1 for 'no band', where banding is an undesirable process delay that arises during the rotogravure printing process. By training ML models on this dataset, you could help identify or predict cases where these process delays are avoidable, thereby improving the efficiency of the printing. For the sake of this exercise, we only consider features 21-39 in the above link, and have removed any samples with missing values in that range. No further processing of the data is required on your behalf. The data has been partitioned into a training and testing set using an 80/20 split. Your models will be trained on just the test set, and accuracy results will be reported on both the training and testing sets.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- Accuracy function
- Report accuracy of the DT model on the training and testing set
- Report accuracy of the Random Forest model on the training and testing set

Imports and Utility Functions:

In [2]:
```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Load the data

Use the `np.load()` function to load "w5-hw1-train.npy" (training data) and "w5-hw1-test.npy" (testing data). The first 19 columns of each are the features. The last column is the label

In [10]:
```
# YOUR CODE GOES HERE
train = np.load("data/w5-hw1-train.npy")
test = np.load("data/w5-hw1-test.npy")
train_x = train[:,:19]
print(train_x)
train_y = train[:,19]
print(train_y)
```

```
test_x = test[:,:19]
test_y = test[:,19]
```

```
[[ 40.     43.      0.267 ...  40.    103.22  100.    ]
 [ 45.     58.      0.2   ...  40.    103.125 100.    ]
 [ 32.5    47.      0.267 ...  40.    100.    100.    ]
 ...
 [ 50.     62.      0.4   ...  40.    110.    100.    ]
 [ 40.     47.      0.333 ...  40.    100.    100.    ]
 [ 65.     65.      0.4   ...  35.    100.    100.    ]]
[1. 1. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1.
 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0.
 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1.
 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1.
 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1.
 0. 0. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0.
 1. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0.
 1. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1.
 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 1.
 1. 0. 0.]
```

Write an accuracy function

Write a function `accuracy(pred,label)` that takes in the models prediction, and returns the percentage of predictions that match the corresponding labels.

In [6]:
```
# YOUR CODE GOES HERE
def accuracy(pred,label):
    return 100*np.sum(pred == label)/len(label)
```

Train a decision tree model

Train a decision tree using `DecisionTreeClassifier()` with a `max_depth` of 10 and using a `random_state` of 0 to ensure repeatable results. Print the accuracy of the model on both the training and testing sets.

In [13]:
```
# YOUR CODE GOES HERE
tree = DecisionTreeClassifier(max_depth = 10, random_state = 0)
tree.fit(train_x,train_y)
pred_train = tree.predict(train_x)
acc_train = accuracy(pred_train,train_y)
print("Accuracy of the train data is: ",acc_train,"%")

pred_test = tree.predict(test_x)
acc_test = accuracy(pred_test,test_y)
acc_test = accuracy(pred_test,test_y)
print("Accuracy of the test data is: ",acc_test,"%")
```

```
Accuracy of the train data is:  93.12714776632302 %
Accuracy of the test data is:  65.75342465753425 %
```

Train a random forest model

Train a random forest model using `RandomForestClassifier()` with a `max_depth` of 10, a `n_estimators` of 100, and using a random state of `0` to ensure repeatable results. Print the accuracy of the model on both the training and testing sets.

In [15]:
```
# YOUR CODE GOES HERE
forest = RandomForestClassifier(max_depth = 10, n_estimators = 100, random_state = 0)
forest.fit(train_x,train_y)
pred_train = forest.predict(train_x)
acc_train = accuracy(pred_train,train_y)
print("Accuracy of the train data is: ",acc_train,"%")

pred_test = forest.predict(test_x)
acc_test = accuracy(pred_test,test_y)
acc_test = accuracy(pred_test,test_y)
print("Accuracy of the test data is: ",acc_test,"%")
```

```
Accuracy of the train data is:  100.0 %
Accuracy of the test data is:  82.1917808219178 %
```

Discuss the performance of the models

Compare the training and testing accuracy of the two models, and explain why the random forest model is advantageous compared to a standard decision tree model

The random forest model gave us better accuracy than the standard decision tree model. The random forest model trains multiple parallel trees therefore providing better generalization of the data. It also uses subsets of features which helps in de-correlating the data better. This improves the performance of the model and tehrefore the model doesn't overfit.

In [ ]:
```
```